Assignment 8

Q 1 Adjacency matrix

We use an n x n matrix to represent a graph of n nodes.

∴ For every node, we have a row and column of n elements,

To check neighbours of a particular node, we find all the elements where we see a '1' instead of '0' (or any representation) istead of 0s and 1s. Thus, we traverse the entire row/column of length 'n'. We do this 'n' times.

Other operations of stack are constant.

∴ Traversal time is $O(n^2)$

Adjacency List

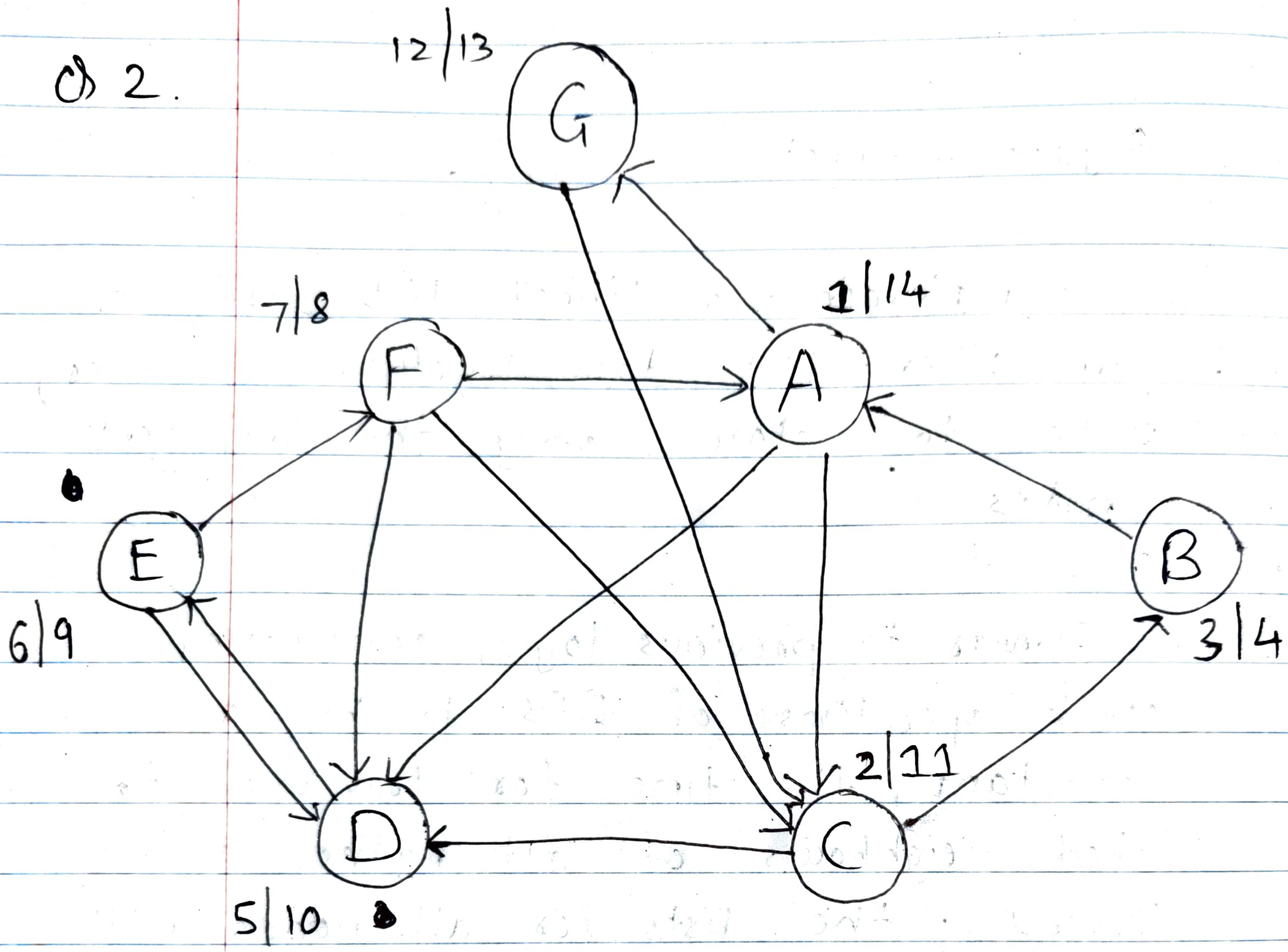
We ~~a~~ maintain a linked list /array or any other list data structure for every node to store only the adjacent nodes.

∴ Similar to previous logic, considering stack operations of DFS to be a constant, the time for traversal to check neighbours of all nodes, i.e., traversing the lists for all nodes will give the time complexity.

~~Number~~ Length of ~~linked~~ list is variable, but we can get the number of nodes we traverse in total as an expression of $V + E$ where V is the number of nodes and e is the number of edges in the graph.

~~Ero~~ DFS time is $O(n + E)$

Q 2.



Assuming start node to be vertex A, the numbers on the left are discover times and the numbers on the right are finishing times for each node.

Q3, From the white-part theorem, we know that during DFS traversal for a pair of two nodes ~~who~~ which are ancestor - descendant, the discovery time of ancestor is less than that of the descendant because the ancestor will be discovered first.

Similarly, the finishing time will be opposite, i.e, the finishing time of descendant will be less than the finishing time of the ancestor.

∴ We write the pseudo code ~~as~~ to calculate the discovery and finish times of u and v by ~~the~~ DFS traversal ~~as~~ : and then to predict our answer as:

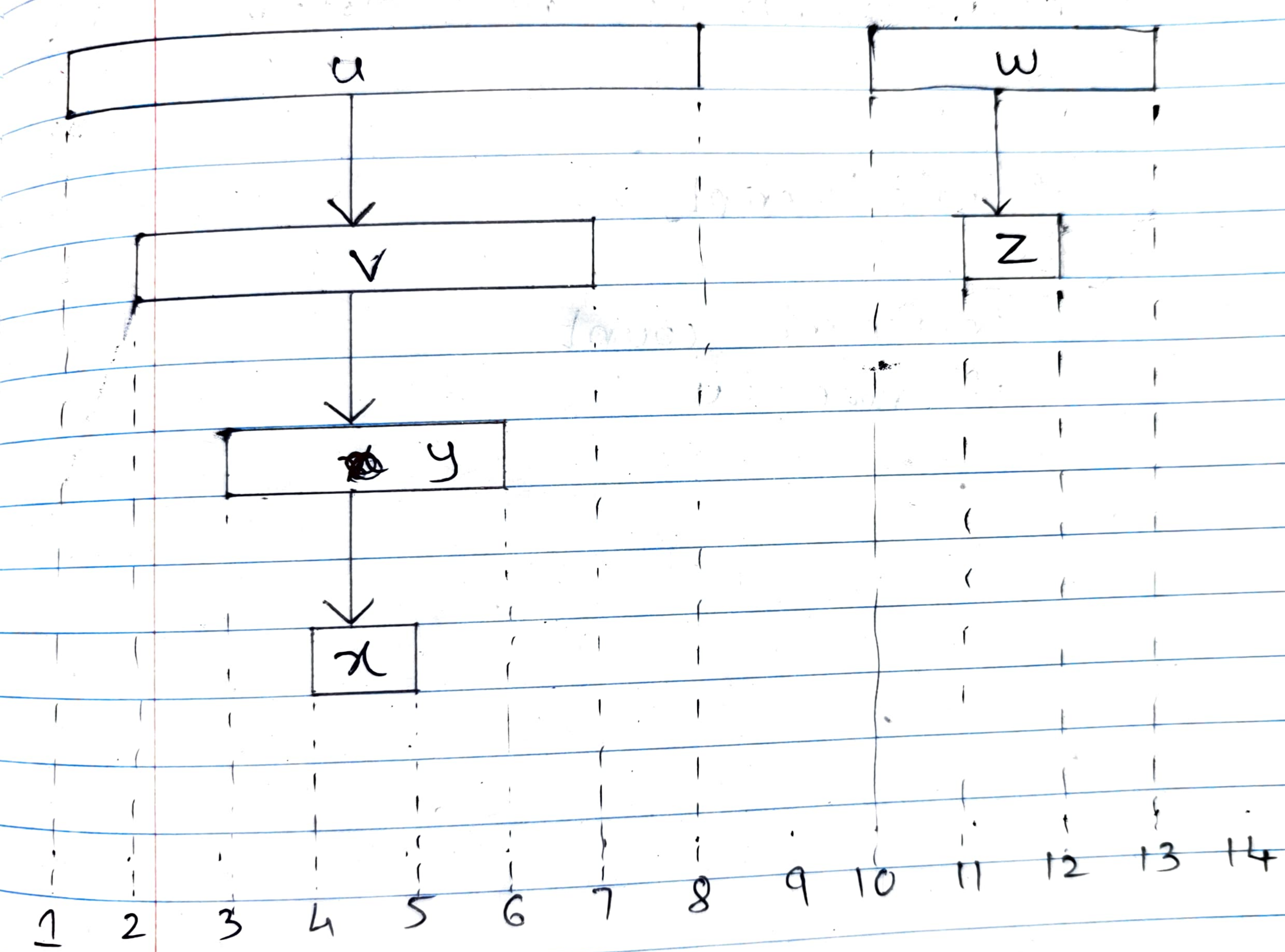
Traversal to get times

```
checkAncDes( u, v, d_timeU, f_timeU, d_timeV,
                f_timeV, count )
{
  if  node is not empty
  {
     count ++
     if  node == u
     {
        if  d_timeU is never initialized
        {
           d_timeU = count
        }
        else  f_timeU = count
        checkAscDes(node.left, u, v, d_timeU,
              f_timeU, d_timeV, f_timeV, count)
        checkAscDes(node_right, u, v, d_timeU,
              f_timeU, d_timeV, f_timeV, count)
     }
     else if(node == v)
     {
        if  d_timeV is never initialized
              d_timeV = count
        else
              f_timeV = count
```

```
check AscDes (node.left, u, v, d_timeU,
        f_timeU, d_timeV, f_timeV, count)
check AscDes ( for right subtree )
}
}

A Analyse the values
findAnswer (d_timeU, f_timeU, d_timeV,
        f_timeV )
{
  if ( d_timeU < d_timeV &&
          f_timeU > f_timeU )
  {
     " U is ancestor of V "
  }
  else if( d_timeU > d_timeV &&
          f_timeU < f_timeV )
  {
     " V is ancestor of U "
  }
  else " error "
}
```

Q4)



(u (v (y (x x) y) v) u) (w (z z) w)